Switching Neural Networks: A New Connectionist Model for Classification

Conference Paper *in* Lecture Notes in Computer Science · January 2005

DOI: 10.1007/11731177_4 · Source: DBLP

1 author:

Marco Muselli
Italian National Research Council
125 PUBLICATIONS **2,000** CITATIONS

SEE PROFILE

Switching Neural Networks: A New Connectionist Model for Classification

Marco Muselli

Istituto di Elettronica e di Ingegneria dell'Informazione e delle Telecomunicazioni
Consiglio Nazionale delle Ricerche
via De Marini, 6 - 16149 Genova, Italy
Email: marco.muselli@ieiit.cnr.it

Abstract. A new connectionist model, called *Switching Neural Network* (*SNN*), for the solution of classification problems is presented. SNN includes a first layer containing a particular kind of A/D converters, called *latticizers*, that suitably transform input vectors into binary strings. Then, the subsequent two layers of an SNN realize a positive Boolean function that solve in a lattice domain the original classification problem. Every function realized by an SNN can be written in terms of intelligible rules. Training can be performed by adopting a proper method for positive Boolean function reconstruction, called *Shadow Clustering* (*SC*). Simulation results obtained on the StatLog benchmark show the good quality of the SNNs trained with SC.

1 Introduction

Any classification problem can be viewed as an optimization problem, where a proper functional, i.e. the probability of error, has to be minimized by choosing the best classifier g inside a given collection Γ. In this sense any technique for the solution of classification problems must provide for two different actions: the class Γ of decision functions must be suitably determined (*model selection*) and the best classifier in Γ must be searched for (*training phase*).

General theoretical results advise us against choosing a too large set Γ; in fact, with this choice it is likely to incur in the problem of *overfitting*: the training phase generates a decision function $g \in \Gamma$ that performs well on the examples of the training set, but scores a high number of misclassifications on the remaining points of the input domain. On the other hand, if a too small set Γ is considered, it is impossible to obtain a sufficiently low number of errors in the training set.

A possible approach consists in choosing initially a large Γ, leaving to the learning algorithm the task of retrieving a classifier g which is enough simple, according to some index of complexity, and behaves well on the training set. This is the approach followed by support vector machines [1], where a regularization constant controls the trade-off between the complexity of the resulting decision function and the number of errors scored by it on the available training set.

In a similar way, we will introduce in the following sections a new connectionist model, called *Switching Neural Network (SNN)*, which is sufficiently rich to

approximate within an arbitrary precision any measurable function. As described later, this connectionist model presents some interesting properties, among which the possibility of allowing a precise description of classifiers $g \in \Gamma$ in terms of intelligible rules.

A proper learning algorithm, called *Shadow Clustering* (*SC*), can be adopted to search for the simplest SNN that ensures a sufficiently low number of errors on the training set. Preliminary results show that SNNs trained by SC can achieve generalization errors comparable with those of best machine learning techniques. Due to space limitation, details of SC are not presented here, but can be found elsewhere [2].

2 Model selection

Consider a general binary classification problem, where d-dimensional patterns $\boldsymbol{x} \in X \subset \mathbb{R}^d$ are to be assigned to one of two possible classes, labeled by the values of a Boolean output $y \in \{0,1\}$. According to possible situations in real world problems, the type of the components x_i, $i = 1,\ldots,d$, can be either *continuous ordered*, when x_i belongs to a subset of $\mathbb{R}$, or *discrete ordered*, when x_i assumes values inside a finite ordered set, or *nominal*, when x_i can assume values inside a finite set, where no ordering is defined.

Denote with I_m the set $\{1,2,\ldots,m\}$ of the first m positive integers. Without loss of generality, I_m (for a proper value of m) can be regarded as the domain of any discrete ordered or nominal variable. In the first case, I_m is viewed as a chain, whereas in the latter no ordering is assumed to be defined on it.

Now, consider the Boolean lattice $\{0,1\}^n$, equipped with the well known binary operations '$+$' (*logical sum* or OR) and '$\cdot$' (*logical product* or AND). According to the standard partial ordering on $\{0,1\}^n$, a Boolean function $f : \{0,1\}^n \to \{0,1\}$ is called *positive* if $\boldsymbol{u} \leq \boldsymbol{v}$ implies $f(\boldsymbol{u}) \leq f(\boldsymbol{v})$ for every $\boldsymbol{u}, \boldsymbol{v} \in \{0,1\}^n$.

A recent theoretical result [3] asserts that positive Boolean functions are universal approximators, i.e. they can approximate arbitrarily well every measurable function $g : \mathbb{R}^d \to \{0,1\}$. Denote with Q_n^l the subset of $\{0,1\}^n$ containing the strings of n bits having exactly l values 1 inside them. A possible procedure for finding the positive Boolean function f that approximates a given classifier g is based on the following three steps:

1. (*Discretization*) For every ordered input x_i, determine a finite partition $\mathcal{B}_i$ of its domain X_i such that a function $\hat{g}$ can be found, which approximates g on X within the desired precision and assumes a constant value on every set $B \in \mathcal{B}$, where $\mathcal{B} = \{\prod_{i=1}^{d} B_i \,:\, B_i \in \mathcal{B}_i,\, i = 1,\ldots,d\}$.
2. (*Latticization*) By employing a proper function $\boldsymbol{\varphi}$, map the points of the domain X into the strings of Q_n^l, so that $\boldsymbol{\varphi}(\boldsymbol{x}) \neq \boldsymbol{\varphi}(\boldsymbol{x}')$ only if $\boldsymbol{x} \in B$ and $\boldsymbol{x}' \in B'$, being B and B' two different sets in $\mathcal{B}$.
3. (*Positive Boolean function synthesis*) Select a positive Boolean function f such that $f(\boldsymbol{\varphi}(\boldsymbol{x})) = \hat{g}(\boldsymbol{x})$ for every $\boldsymbol{x} \in X$.

2.1 Discretization

Since the exact behavior of the decision function g is not known, the approximating classifier $\hat{g}$ and the partition $\mathcal{B}$ have to be inferred from the examples $(\boldsymbol{x}_j, y_j)$ of the training set S. Every set $B_i \in \mathcal{B}_i$ must be enough large to include the component x_{ji} of some point $\boldsymbol{x}_j$ in S. Nevertheless, the resulting partition $\mathcal{B}$ must be enough fine to capture the actual complexity of the function g.

Several different discretization methods for binary classification problems have been proposed in the literature [4–6]. Usually, for each ordered input x_i a set of $m_i - 1$ consecutive values $r_{i1} < r_{i2} < \cdots < r_{i,m_i-1}$ is generated and the partition $\mathcal{B}_i$ is formed by the m_i sets $X_i \cap R_{ik}$, where $R_{i1} = (-\infty, r_{i1}]$, $R_{i2} = (r_{i1}, r_{i2}]$, ..., $R_{i,m_i-1} = (r_{i,m_i-2}, r_{i,m_i-1}]$, $R_{im_i} = (r_{i,m_i-1}, +\infty)$. Excellent results have been obtained with the algorithm Chi2 [5], which employ the χ^2 statistic to decide the position of the points r_{ik}, $k = 1, \ldots, m_i - 1$, and with the technique EntMDL [4], which adopts entropy estimates to achieve the same goal. An alternative promising approach is offered by the method used in the LAD system [6]: in this case an integer programming problem is solved to obtain optimal values for the cutoffs r_{ik}.

By applying a procedure of this kind, the discretization task defines for each ordered input x_i a mapping $\psi_i : X_i \to I_{m_i}$, where $\psi_i(z) = k$ if and only if $z \in R_{ik}$. If we assume that ψ_i is the identity function with $m_i = |X_i|$ when x_i is a nominal variable, the approximating function $\hat{g}$ is uniquely determined by a discrete function $h : I \to \{0, 1\}$, defined as $h(\boldsymbol{\psi}(\boldsymbol{x})) = \hat{g}(\boldsymbol{x})$, where $I = \prod_{i=1}^{d} I_{m_i}$ and $\boldsymbol{\psi}(\boldsymbol{x})$ is the mapping from X to I, whose ith component is given by $\psi_i(x_i)$.

2.2 Latticization

The function $\boldsymbol{\psi}$ provides a mapping from the domain X onto the set $I = \prod_{i=1}^{d} I_{m_i}$, such that $\boldsymbol{\psi}(\boldsymbol{x}) = \boldsymbol{\psi}(\boldsymbol{x}')$ if $\boldsymbol{x}$ and $\boldsymbol{x}'$ belong to the same set $B \in \mathcal{B}$, whereas $\boldsymbol{\psi}(\boldsymbol{x}) \neq \boldsymbol{\psi}(\boldsymbol{x}')$ only if $\boldsymbol{x} \in B$ and $\boldsymbol{x}' \in B'$, being B and B' two different sets in $\mathcal{B}$. Consequently, the 1-1 function $\boldsymbol{\varphi}$ from X to Q_n^l, required in the latticization step, can be simply determined by defining a proper 1-1 function $\boldsymbol{\beta} : I \to Q_n^l$. In this way, $\boldsymbol{\varphi}(\boldsymbol{x}) = \boldsymbol{\beta}(\boldsymbol{\psi}(\boldsymbol{x}))$ for every $\boldsymbol{x} \in X$.

A possible way of constructing the function $\boldsymbol{\beta}$ is to define in a proper way d mappings $\boldsymbol{\beta}_i : I_{m_i} \to Q_{n_i}^{l_i}$; then, the binary string $\boldsymbol{\beta}(\boldsymbol{u})$ for an integer vector $\boldsymbol{u} \in I$ is obtained by concatenating the strings $\boldsymbol{\beta}_i(u_i)$ for $i = 1, \ldots, d$. It can be shown [3] that a good choice for $\boldsymbol{\beta}_i$ is the *inverse only one coding*, which maps an integer $u_i \in I_{m_i}$ into the binary string $\boldsymbol{z}_i \in Q_{m_i}^{m_i-1}$ having $z_{ik} = 0$ if and only if $u_i = k$. In fact, this coding is both an isometry and a full order-preserving mapping (when x_i is an ordered input). These properties characterizes also the mapping $\boldsymbol{\beta}$ if the inverse only one coding is adopted for all the $\boldsymbol{\beta}_i$.

Since $\boldsymbol{\varphi}(\boldsymbol{x})$ is obtained by the concatenation of d binary strings $\boldsymbol{\varphi}_i(x_i)$, if the discretization task has produced for each ordered input x_i a set of $m_i - 1$ cutoffs r_{ik}, as described in the previous subsection, the kth bit of $\boldsymbol{\varphi}_i(x_i)$ assumes value 0 if and only if $x_i \in R_{ik}$. Note that $x_i \in R_{ik}$ if and only if x_i exceeds the cutoff $r_{i,k-1}$ (if $k > 1$) and is lower than the subsequent cutoff r_{ik} (if $k < m_i$). On the

other hand, if x_i is a nominal input the kth bit of $\boldsymbol{\varphi}_i(x_i)$ assumes value 0 if and only if $x_i = k$.

Thus, the mapping $\boldsymbol{\varphi}_i$ can be implemented by a simple device that receives in input the value x_i and compares it with a sequence of integer or real numbers. This device will be called *latticizer*; it produces m_i binary outputs, but only one of them can assume the value 0. The whole mapping $\boldsymbol{\varphi}$ is realized by a parallel of d latticizer, each of which is associated with a different input x_i.

2.3 Switching Neural Networks

After the discretization and the latticization steps we have transformed the training set $S = \{(\boldsymbol{x}_j, y_j), j = 1, \ldots, s\}$ into a collection of examples $S' = \{(\boldsymbol{z}_j, y_j), j = 1, \ldots, s\}$, where each $\boldsymbol{z}_j = \boldsymbol{\varphi}(\boldsymbol{x}_j)$ is a binary string in the Boolean lattice $\{0,1\}^m$ for a proper value of m. The original binary classification problem can then be solved by retrieving a positive Boolean function $f(\boldsymbol{z})$ such that $\hat{g}(\boldsymbol{x}) = f(\boldsymbol{\varphi}(\boldsymbol{x}))$ minimizes the probability of misclassifying a pattern $\boldsymbol{x} \in X$.

This target can be pursued by adopting a proper technique, named *Shadow Clustering* (*SC*) [2], which is able to construct a positive Boolean function f that generalizes well starting from the examples contained in S'. To our best knowledge, SC is the first method of this kind. It adopts an overall strategy similar to Hamming Clustering [7], successfully employed in the solution of binary classification problems. A detailed description of the approach followed by SC for the synthesis of positive Boolean functions is presented in [2].

If $\boldsymbol{a} \in \{0,1\}^m$ is a binary string with length m, let $P(\boldsymbol{a})$ be the subset of I_m including the indexes i for which $a_i = 1$. A positive Boolean function can always be written in the following *Positive Disjunctive Normal Form* (*PDNF*):

$$f(\boldsymbol{z}) = \bigvee_{\boldsymbol{a} \in A} \bigwedge_{j \in P(\boldsymbol{a})} a_j \tag{1}$$

where A is an *antichain* of the Boolean lattice $\{0,1\}^m$, i.e. a collection of binary strings such that if $\boldsymbol{a}, \boldsymbol{a}' \in A$ neither $\boldsymbol{a} < \boldsymbol{a}'$ nor $\boldsymbol{a}' < \boldsymbol{a}$. The symbol $\bigvee$ (resp. $\bigwedge$) in (1) denotes a logical sum (resp. product) among the terms identified by the subscript. In particular, $\bigwedge_{j \in P(\boldsymbol{a})} z_j$ is an *implicant* for the function f; however, when no confusion arises, the term implicant will also be used to denote the corresponding binary string $\boldsymbol{a} \in A$.

The execution of SC produces the antichain A to be employed in (1) for obtaining the PDNF of the desired positive Boolean function f. This expression can be readily implemented on a two-level digital circuits including only AND, OR ports. It should be observed that only the values 1 in a binary string $\boldsymbol{a} \in A$ give rise to an incoming connection in the corresponding AND port. Thus, values 0 in $\boldsymbol{a}$ behave as don't care symbols for the associated implicant.

The approximating function $\hat{g}(\boldsymbol{x})$ that solves our binary classification problem is then given by the composition $f(\boldsymbol{\varphi}(\boldsymbol{x}))$ of the positive Boolean function f with the mapping $\boldsymbol{\varphi}$ produced by discretization and latticization steps. The device implementing $\hat{g}(\boldsymbol{x})$ is shown in Fig. 1.

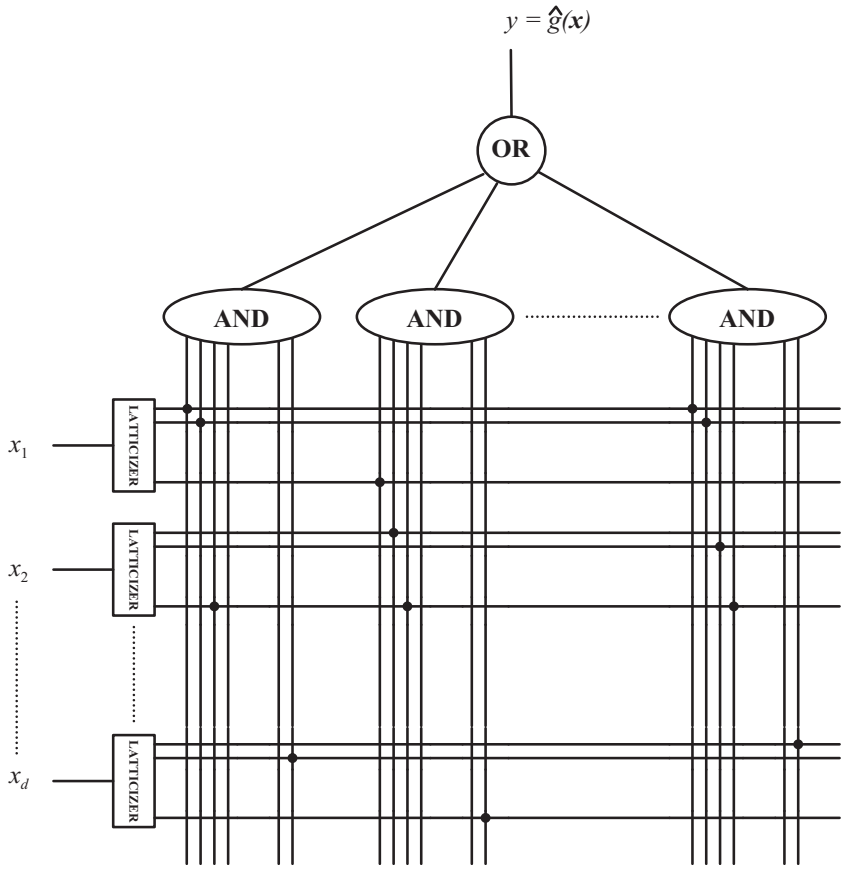

Fig. 1. Schema of a Switching Neural Network.

It can be considered as a three layer feedforward neural network, where the first layer is responsible of the realization of the binary mapping $\boldsymbol{\varphi}(\boldsymbol{x})$, while the other two realize the expression (1) for the positive Boolean function f. Every AND port in the second layer is connected only to some of the outputs leaving the latticizers; they correspond to values 1 in the associated implicant. These connections are represented by bold circles in Fig. 1 and can be viewed as switches that establish links between the inputs of the AND ports and the outputs of the latticizers. For this reason the connectionist model shown in Fig. 1 is called *Switching Neural Network* (*SNN*).

It is interesting to observe that, unlike standard neural networks, SNNs do not involve weights; furthermore, signals traveling on them have only one level. Thus, it should be concluded that the behavior of the function $\hat{g}(\boldsymbol{x})$ is entirely memorized in the architecture of the SNN (connections and switches). This is not a limitation, since it has been shown that SNN are universal approximators.

Every implicant $\boldsymbol{a} \in A$ generated by SC can be translated into an intelligible rule in the **if-then** form underlying the classification problem at hand. Consider the substrings $\boldsymbol{a}_i$ of $\boldsymbol{a}$ that are associated with the ith input x_i to the network. The logical product $\bigwedge_{j \in P(\boldsymbol{a})} z_j$ gives output 1 only if the binary string $\boldsymbol{z} = \boldsymbol{\varphi}(\boldsymbol{x})$ presents a value 1 in all the positions where $\boldsymbol{a}_i$ has value 1.

If x_i is an ordered variable, the execution of SC always generates binary strings $\boldsymbol{a}_i$ containing a single sequence of consecutive values 0, i.e. a run of 0. If this run begins at the $(j+1)$th position and finishes at the kth bit of $\boldsymbol{a}_i$, the logical product $\bigwedge_{j \in P(\boldsymbol{a})} z_j$ can give output 1 only if $r_{ij} < x_i \leq r_{ik}$. In the particular case where the run of 0 begins at the first position (resp. finishes at the last position), the condition becomes $x_i \leq r_{ik}$ (resp. $x_i > r_{ij}$).

As an example, suppose that an ordered variable x_i has been discretized by using the four cutoffs 0.1, 0.25, 0.3, 0.5. If the implicant $\boldsymbol{a}$ with $\boldsymbol{a}_i = 10011$ has been produced by SC, the condition $0.1 < x_i \leq 0.3$ has to be included in the **if** part of the **if-then** rule associated with $\boldsymbol{a}$.

On the other hand, if x_i is a nominal variable the portion $\boldsymbol{a}_i$ of an implicant $\boldsymbol{a}$ gives rise to the condition $x_i \in \bigcup_{k \in I_{m_i} \setminus P(\boldsymbol{a}_i)} \{k\}$. Again, if the implicant $\boldsymbol{a}$ with $\boldsymbol{a}_i = 01101$ has been produced by SC, the condition $x_i \in \{1, 4\}$ has to be included in the **if-then** rule associated with $\boldsymbol{a}$. In any case, if the binary string $\boldsymbol{a}_i$ contains only values 0, the input x_i will not be considered in the rule for $\boldsymbol{a}$.

Thus, it follows that every implicant $\boldsymbol{a}$ gives rise to an **if-then** rule, having in its **if** part a conjunction of the conditions obtained from the substrings $\boldsymbol{a}_i$ associated with the d inputs x_i. If all these conditions are verified, the output $y = \hat{g}(\boldsymbol{x})$ will be assigned the value 1. A logical OR connects all the rules obtained in this way for every implicant $\boldsymbol{a}$ in the antichain A produced by SC.

Due to this property, SC (with the addition of discretization and latticization) becomes a rule generation method, being capable of retrieving from the training set some kind of intelligible information about the physical system underlying the binary classification problem at hand.

3 Simulation results

To obtain a preliminary evaluation of performances achieved by SNNs trained with SC, the ten classification problems included in the well-known StatLog benchmark [8] have been considered. In this way the generalization ability and the complexity of resulting SNNs can be compared with those of other machine learning methods, among which rule generation techniques based on decision trees, such as C4.5 [9]. In all these simulations the discretization method adopted in the LAD system [6] has been used to map continuous inputs into binary strings.

The tests contained in the StatLog benchmark present different characteristics that allow to evaluate the behavior of a classification algorithm under several angles. Four problems (Heart, Australian, Diabetes, and German) presents a binary output, thus permitting a direct application of the SNN approach, as

described in the previous section. However, two of them (Heart and German) adopts a specific cost matrix to weight misclassified patterns.

The remaining six tests concern multiclass problems, which have been split into a sequence of binary classification problems by constructing a separate set of implicants for each output value. The class of an new pattern $\boldsymbol{x}$ has then been chosen by adopting the criteria introduced in [10], which performs a weighted sum of perfect matching rules (those verified by $\boldsymbol{x}$), having weight 1, and almost matching rules (those verified by $\boldsymbol{x}$, except for one condition), which is assigned the weight 0.1.

The complexity of an SNN is measured through the number of AND ports in the second layer (corresponding to the number of intelligible rules) and the average number of conditions in the **if** part of a rule. Tab. 1 presents the results obtained. Accuracy and complexity of resulting SNNs are compared with those of rulesets produced by C4.5. In the same table is also reported the best generalization error reported in the StatLog report [8] for each problem, together with the rank scored by SNN when its generalization error is inserted into the list of available results.

Table 1. Generalization error and complexity of SNN, compared with C4.5 and with other methods, on the StatLog benchmark.

Test	Generalization error				# Rules		# Conditions	
Problem	SNN	C4.5	Best	Rank	SNN	C4.5	SNN	C4.5
Heart	0.393	0.781	0.374	2	24.3	11.4	5.03	2.68
Australian	0.125	0.155	0.131	1	26.4	11.5	5.55	2.76
Diabetes	0.250	0.270	0.223	8	73.8	9.4	4.61	2.58
Vehicle	0.278	0.266	0.150	12	91.2	26.1	5.92	4.03
German	0.716	0.985	0.535	13	95.8	21.1	8.90	2.77
Segment	0.037	0.040	0.030	11	82.8	28.0	4.51	3.94
DNA	0.057	0.076	0.041	3	132.0	34.0	8.99	4.47
Satimage	0.135	0.150	0.094	6	262.0	80.0	7.92	5.41
Letter	0.115	0.132	0.064	5	1532.0	570.0	8.38	7.64
Shuttle	0.0001	0.001	0.0001	1	18.0	20.0	3.17	3.14

Apart from one case (Vehicle) the generalization error scored by SNN is always lower than that obtained by C4.5. On the other hand, the complexity of SNN is considerably higher (except for the Shuttle problem). As a consequence of this greater complexity, the execution time of SNN is significantly higher than that of C4.5. It ranges from 3 sec. (Australian) to a hour (Letter) for the StatLog benchmark, whereas the construction of the set of rules with C4.5 requires at most three minutes.

This behavior depends on the method employed to produce the implicants and the corresponding rules. In SNN every rule separates some patterns of a class from all the example in the training set belonging to other classes; at most a small error is accepted to avoid overfitting. On the contrary, C4.5 creates rules

that are verified by a subset of patterns from different classes. Subsequent rules correct errors performed by previous ones; therefore, rules must be applied in a specific order.

It is interesting to note that in four out of the ten problems SNN achieves one of the first three ranking positions. This points out the quality of the solutions offered by SNN, even if its behavior in dealing with multiclass problems can be improved by properly adapting the SC algorithm to reconstruct in an efficient way positive Boolean functions with several outputs.

References

1. Vapnik, V.N.: Statistical Learning Theory. New York: John Wiley & Sons (1998)
2. Muselli, M., Quarati, A.: Shadow Clustering: A method for monotone Boolean function synthesis. Technical Report IEIIT/GE/2/03, Istituto di Elettronica e di Ingegneria dell'Informazione e delle Telecomunicazioni, Consiglio Nazionale delle Ricerche (2003)
3. Muselli, M.: Approximation properties of positive Boolean functions. *Submitted to the* WIRN '05 - XVI Italian Workshop on Neural Networks (2005)
4. Kohavi, R., Sahami, M.: Error-based and entropy-based discretization of continuous features. In: Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (1996) 114–119
5. Liu, H., Setiono, R.: Feature selection via discretization. IEEE Transactions on Knowledge and Data Engineering **9** (1997) 642–645
6. Boros, E., Hammer, P.L., Ibaraki, T., Kogan, A., Mayoraz, E., Muchnik, I.: An Implementation of Logical Analysis of Data. IEEE Transactions on Knowledge and Data Engineering **12** (2000) 292–306
7. Muselli, M., Liberati, D.: Binary rule generation via Hamming Clustering. IEEE Transactions on Knowledge and Data Engineering **14** (2002) 1258–1268
8. Michie, D., Spiegelhalter, D., Taylor, C., eds.: Machine Learning, Neural, and Statistical Classification, London: Ellis-Horwood (1994)
9. Quinlan, J.R.: C4.5: Programs for Machine Learning. San Francisco: Morgan Kaufmann (1994)
10. Hong, S.J.: R-MINI: An Iterative Approach for Generating Minimal Rules from Examples. IEEE Transactions on Knowledge and Data Engineering **9** (1997) 709–717